PA2 Document

Written by Hyeonho Shin, 20203344, Master Course, 010-5136-6003

Summary

My homework is Resnet50-based, which is the well-known Net in image-classification area. But for better operating in our set I used more tricks (Adaptive learning rate, small batchsize, modify ResNet50, and DataAugumentation)

The simplest way is to execute **'ForTA.py'**. It loads the weights what I trained using the various technique.

How to use

Where are preprocessing and classifying functions?

All preprocessing steps are included in tanukiDataAug.py. Trainers call them before learning.

For faster learning, I seperated Preprocessing function as ‘gen_aug.py’. It enhances, take ROI of images and save the results on your disk, ‘./aug_abcde’. So, in this process, all you need to is

1. Run gen_aug.py to preprocess.
2. Run tanukiChar_trainer_with_gen_aug.py.
3. Take the model having the best accuracy from step 2.
4. Run ForTA.py for evaluating. If you designate the input file name, type ‘python ForTA.py -i input_file.pth’

But if you want to see the preprocess to get more certainty, you can use tanuki_trainer only. It follows Pytorch’s transform class. Whenever you read batch data from disk, it preprocesses the batch size. So, it’s more easy to see but I do not recommend to execute it. Because it takes too much time. I attached the preprocess program to just help TA to score my HW. In summary, using tanukiChar_trainer.py, what you need to do is…

1. Run tanukiChar_trainer.py
2. Take the model having the best accuracy from step 2.
3. Run ForTA.py for evaluating. If you designate the input file name, type ‘python ForTA.py -i input_file.pth’

For scoring

Excecute **ForTA.py**. There is no need to use any arguments. It reads **"FtanukiCharNet.pth"** weight file on the same folder.

Requisites

- Pytorch (1.7.0), (in 1.3.1 it will make error because save method is different.)
- torchvision
- tanukiCharNet.py, tanukiDataAug.py in same folder.

Design

Abstract

For overcoming the small dataset, I adopted 4 techniques.

1. Augumentation.

My codes generate randomly rotated, shifted, and resized figures without loss. In short, different to the common transform functions, the whole character is not destroyed. It can be achieved by boundary extraction. My code extracts the ROI (Region of Interest) using bounding box function. To make the function well-operated, I applied bilateralFilter and adaptive thresholding in prior. And then for denosing, I applied randomly constrast increasing. From ROI, I resized ROI randomly, but using LANCZOS for lower reszing noise. And then translate it in only 350*350 array.

Because of the speed problem, I implemented this by two methods. First one is simple, I made custom transform function for Pytorch. It looks nice and easy to use.(It is implemented in *tanukiChar_trainer.py*) But data loading is too slow because numpy cannot support multiprocessing. For solving it, I just implemented the program to save augumented image in HDD first,"*gen_aug.py*". And then train it with "*tanukiChar_trainer_with_gen_aug.py*". It is faster.

2. Small batch size

By some papers, they argued that generalization performance increase in more various learning rate under small batch size. It recommends batch size as 8.

3. Adaptive learning rate

At higher learning rate, we can get more faster learning. But there can be divergence or vibration. So I decrease the learning rate epoch by epoch.

4. Modify ResNet50

Sadly, ResNet50 only supports 224*224 images with 1000 classes. So I adopted AdaptiveAvgPool2d layer and modified the final FC layer to 5 without downscaling of input images.

Results

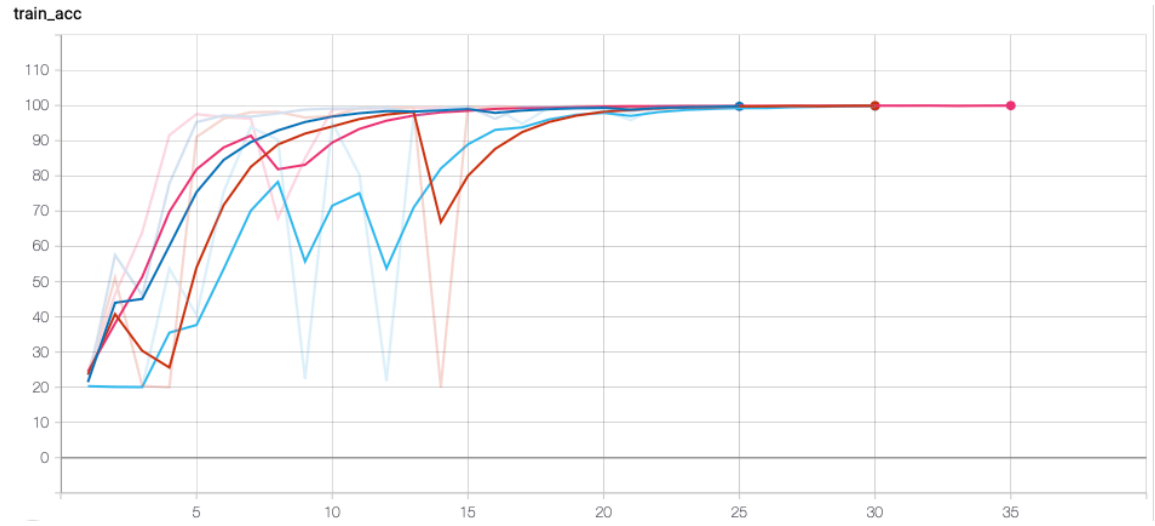
Summary

I choosed my model with epochs 33, batch size 8, learning rate 0.002. It was the best result in searching space batch size: 8 or 16, learning rate 0.001 or 0.002

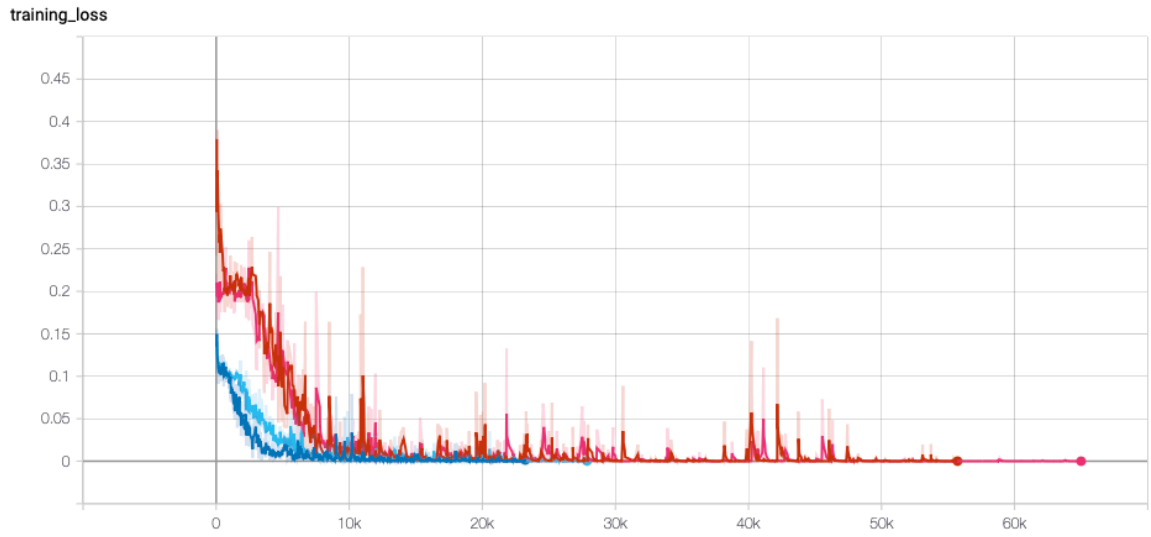
Gray lines are real value, and strong lines are polyfitted value.

Batch size\Learning rate	0.001	0.002
8	Red	Claret(자홍)
16	Blue	Sky blue

Training Accuracy (by epochs)



Training loss (by iter)



Validation Accuray (by epochs)